2

AD-A247 334

TATION PAGE

Form Approved
OPM No. 0704-0188

Pub
nee
Hea
Man

ur per response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington fferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED Final: 12 Oct 1991 to 01 Jun 1993

4. TITLE AND SUBTITLE
Validation Summary Report: Aitech Defense Systems Inc., AI-ADA/96K, Version 3.0, Sun-4/330 under SunOS 4.1.1(Host) to DSP96002 ADS board (bare machine)(Target), 911012W1.11225

5. FUNDING NUMBERS

6. AUTHOR(S)
Wright-Patterson AFB, Dayton, OH
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ada Validation Facility, Language Control Facility ASD/SCEL
Bldg. 676, Rm 135
Wright-Patterson AFB, Dayton, OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER
AVF-VSR-507.0292

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Pentagon, Rm 3E114
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Aitech Defense Systems Inc., AI-ADA/96K, Version 3.0, Wright-Patterson AFB, Sun-4/330 under SunOS 4.1.1(Host) to DSP96002 ADS board (bare machine)(Target), ACVC 1.11.

14. SUBJECT TERMS
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION UNCLASSIFED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT

AVF Control Number: AVF-VSR-507.0292
4 February 1992
91-06-21-AIT

Ada COMPILER VALIDATION SUMMARY REPORT:

Certificate Number: 911012W1.11225
Aitech Defense Systems Inc.
AI-ADA/96K, Version 3.0
Sun-4/330 under SunOS 4.1.1 => DSP96002 ADS board (bare machine)

Prepared By:
Ada_Validation_Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 12 October 1991.

Compiler Name and Version:	AI-ADA/96K, Version 3.0
Host Computer System:	Sun-4/330 under SunOS 4.1.1
Target Computer System:	DSP96002 ADS board (bare machine)
Customer Agreement Number:	91-06-21-AIT

See section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 911012W1.11225 is awarded to Aitech Defense Systems Inc. This certificate expires on 1 June 1993.

This report has been reviewed and is approved.

[illegible]

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

fo [illegible]

Ada Validation Organization
Director, Computer and Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

John P Solomond

Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301

aitech

DECLARATION OF CONFORMANCE

Customer:	AITECH Defense Systems Inc.
Ada Validation Facility:	ASD/SCEL, Wright-Patterson AFB
ACVC Version	1.11
Ada Implementation:	
Compiler Name and Version:	AI-ADA/96K Version 3.0
Host Computer System:	Sun-4/330 SunOS 4.1.1
Target Computer System:	DSP96002 ADS Board Bare Machine

Customer's Declaration

I, the undersigned, representing AITECH Defense Systems, declare that AITECH Defense Systems has no knowledge of deliberate deviations from Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration.

[illegible]

Gabriel Leemor
AITECH Defense Systems Inc.
3080 Olcott St., Suite 105A
Santa Clara, CA 95054

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Computer and Software Engineering Division
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311-1772

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values — for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1), and possibly removing some inapplicable tests (see section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.

Conformity	Fulfillment by a product, process, or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
LRM	The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>."
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 2 August 1991.

E28005C	B28006C	C32203A	C34006D	C35508I	C35508J
C35508M	C35508N	C35702A	C35702B	B41308B	C43004A
C45114A	C45346A	C45612A	C45612B	C45612C	C45651A
C46022A	B49008A	B49008B	A74006A	C74308A	B83022B
B83022H	B83025B	B83025D	C83026A	B83026B	C83041A
B85001L	C86001F	C94021A	C97116A	C98003B	BA2011A
CB7001A	CB7001B	CB7004A	CC1223A	BC1226A	CC1226B
BC3009B	BD1B02B	BD1B06A	AD1B08A	BD2A02A	CD2A21E
CD2A23E	CD2A32A	CD2A41A	CD2A41E	CD2A87A	CD2B15C
BD3006A	BD4008A	CD4022A	CD4022D	CD4024B	CD4024C
CD4024D	CD4031A	CD4051D	CD5111A	CD7004C	ED7005D
CD7005E	AD7006A	CD7006E	AD7201A	AD7201E	CD7204B
AD7206A	BD8002A	BD8004C	CD9005A	CD9005B	CDA201E
CE2107I	CE2117A	CE2117B	CE2119B	CE2205B	CE2405A
CE3111C	CE3116A	CE3118A	CE3411B	CE3412B	CE3607B
CE3607C	CE3607D	CE3812A	CE3814A	CE3902B	

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by the ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

The following 285 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113F..Y (20 tests)	C35705F..Y (20 tests)
C35706F..Y (20 tests)	C35707F..Y (20 tests)
C35708F..Y (20 tests)	C35802F..Z (21 tests)
C45241F..Y (20 tests)	C45321F..Y (20 tests)
C45421F..Y (20 tests)	C45521F..Z (21 tests)
C45524F..Z (21 tests)	C45621F..Z (21 tests)
C45641F..Y (20 tests)	C46012F..Z (21 tests)

The following 21 tests check for the predefined type SHORT_INTEGER; for this implementation, there is no such type:

C35404B	B36105C	C45231B	C45304B	C45411B
C45412B	C45502B	C45503B	C45504B	C45504E
C45611B	C45613B	C45614B	C45631B	C45632B
B52004E	C55B07B	B55B09D	B86001V	C86006D
CD7101E				

The following 20 tests check for the predefined type LONG_INTEGER; for this implementation, there is no such type:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45613C	C45614C	C45631C	C45632C	B52004D
C55B07A	B55B09C	B86001W	C86006C	CD7101F

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER; for this implementation, there is no such type.

C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT; for this implementation, there is no such type.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT; for this implementation, there is no such type.

C45531M..P and C45532M..P (8 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater; for this implementation, MAX_MANTISSA is less than 47.

C45536A, C46013B, C46031B, C46033B, and C46034B contain length clauses that specify values for 'SMALL that are not powers of two or ten; this implementation does not support such values for 'SMALL.

C45624A..B (2 tests) check that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types and the results of various floating-point operations lie outside the range of the base type; for this implementation, MACHINE_OVERFLOWS is TRUE.

C4A013B contains a static universal real expression that exceeds the range of this implementation's largest floating-point type; this expression is rejected by the compiler.

B86001Y uses the name of a predefined fixed-point type other than type DURATION; for this implementation, there is no such type.

C96005B uses values of type DURATION's base type that are outside the range of type DURATION; for this implementation, the ranges are the same.

CA2009A, CA2009C..D (2 tests), CA2009F, and BC3009C instantiate generic units before their bodies are compiled; this implementation requires that the body of a generic unit be compiled before any instantiation of that unit, as allowed by AI-00506. (See Section 2.3.)

LA3004A..B, EA3004C..D, and CA3004E..F (6 tests) check pragma INLINE for procedures and functions; this implementation does not support pragma INLINE.

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support such sizes.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use length clauses to specify non-default sizes for access types; this implementation does not support such sizes.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions; this implementation provides no package MACHINE_CODE.

The following 264 tests check operations on sequential, text, and direct access files; this implementation does not support external files:

CE2102A..C (3)	CE2102G..H (2)	CE2102K	CE2102N..Y (12)
CE2103C..D (2)	CE2104A..D (4)	CE2105A..B (2)	CE2106A..B (2)
CE2107A..H (8)	CE2107L	CE2108A..H (8)	CE2109A..C (3)
CE2110A..D (4)	CE2111A..I (9)	CE2115A..B (2)	CE2120A..B (2)
CE2201A..C (3)	EE2201D..E (2)	CE2201F..N (9)	CE2203A
CE2204A..D (4)	CE2205A	CE2206A	CE2208B
CE2401A..C (3)	EE2401D	CE2401E..F (2)	EE2401G
CE2401H..L (5)	CE2403A	CE2404A..B (2)	CE2405B
CE2406A	CE2407A..B (2)	CE2408A..B (2)	CE2409A..B (2)
CE2410A..B (2)	CE2411A	CE3102A..C (3)	CE3102F..H (3)
CE3102J..K (2)	CE3103A	CE3104A..C (3)	CE3106A..B (2)
CE3107B	CE3108A..B (2)	CE3109A	CE3110A
CE3111A..B (2)	CE3111D..E (2)	CE3112A..D (4)	CE3114A..B (2)
CE3115A	CE3119A	EE3203A	EE3204A
CE3207A	CE3208A	CE3301A	EE3301B
CE3302A	CE3304A	CE3305A	CE3401A
CE3402A	EE3402B	CE3402C..D (2)	CE3403A..C (3)

CE3403E..F (2)	CE3404B..D (3)	CE3405A	EE3405B
CE3405C..D (2)	CE3406A..D (4)	CE3407A..C (3)	CE3408A..C (3)
CE3409A	CE3409C..E (3)	EE3409F	CE3410A
CE3410C..E (3)	EE3410F	CE3411A	CE3411C
CE3412A	EE3412C	CE3413A..C (3)	CE3414A
CE3602A..D (4)	CE3603A	CE3604A..B (2)	CE3605A..E (5)
CE3606A..B (2)	CE3704A..F (6)	CE3704M..O (3)	CE3705A..E (5)
CE3706D	CE3706F..G (2)	CE3804A..P (16)	CE3805A..B (2)
CE3806A..B (2)	CE3806D..E (2)	CE3806G..H (2)	CE3904A..B (2)
CE3905A..C (3)	CE3905L	CE3906A..C (3)	CE3906E..F (2)

CE2103A, CE2103B, and CE3107A use an illegal file name in an attempt to create a file and expect NAME_ERROR to be raised; this implementation does not support external files and so raises USE_ERROR. (See Section 2.3.)

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 23 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

B55A01A	BA1101E	BA3006A	BA3006B	BA3007B	BA3008A
BA3008B	BA3013A	BC2001D	BC2001E		

CA2009A, CA2009C..D (2 tests), CA2009F, and BC3009C were graded inapplicable by Evaluation Modification as directed by the AVO. These tests instantiate generic units before those units' bodies are compiled; this implementation rejects a unit that contains an instantiation of a unit whose body is not in the program library.

BC3204C..D (2 tests) and BC3205C..D (2 tests) were graded passed by Processing Modification as directed by the AVO. These tests check that instantiations of generic units with unconstrained types as generic actual parameters are illegal if the generic bodies contain uses of the types that require a constraint. However, the generic bodies are compiled after the units that contain the instantiations, and this implementation creates a dependence of the instantiating units on the generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic bodies makes the instantiating units obsolete—no errors are detected. The processing of these tests was modified by re-compiling the obsolete units; all intended errors were then detected by the compiler.

AD7203B was graded passed by Test Modification as directed by the AVO. This implementation allocates 16K words of the target memory for task stacks; by default, equal amounts of storage are allocated to all tasks. AD7203B contains 8 tasks, in addition to the environment task; since the environment task requires in excess of 2K words, STORAGE_ERROR is raised when the test is run. The test was modified by adding a 'STORAGE_SIZE length clause for the task type TSK at line 165 to specify an allocation of 1024 storage units (words) for the activation of each task of the type.

CE2103A, CE2103B, and CE3107A were graded inapplicable by Evaluation Modification as directed by the AVO. The tests abort with an unhandled exception when USE_ERROR is raised on the attempt to create an external file. This is acceptable behavior because this implementation does not support external files. (cf. AI-00332).

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For technical and sales information about this Ada implementation, contact:

Uri Gries
Aitech Defense Systems Inc.
3080 Olcott Street, Suite 105A
Santa Clara, CA 95054
(408) 980-6200

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system — if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

a) Total Number of Applicable Tests	3431
b) Total Number of Withdrawn Tests	95
c) Processed Inapplicable Tests	95
d) Non-Processed I/O Tests	264
e) Non-Processed Floating-Point Precision Tests	285
f) Total Number of Inapplicable Tests	644
g) Total Number of Tests for ACVC 1.11	4170

3.3 TEST EXECUTION

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were loaded directly onto the host computer.

The tests were compiled and linked on the host computer system, as appropriate. The executable images were transferred to the target computer system through ethernet and an OPI (Once Protocol Interface) box manufactured by Aitech and run. The results were captured on the host computer system.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

Option/Switch	Effect
-L	Produces list files.
-A	Produces asm files.

Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
$MAX_IN_LEN	240 — Value of V
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"

MACRO PARAMETERS

$MAX_STRING_LITERAL	'"' & (1..V-2 => 'A') & '"'

The following table lists all of the other macro parameters and their respective values.

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	1
$COUNT_LAST	2_147_483_647
$DEFAULT_MEM_SIZE	32768
$DEFAULT_STOR_UNIT	32
$DEFAULT_SYS_NAME	DSP96002
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	16#400#
$ENTRY_ADDRESS1	16#401#
$ENTRY_ADDRESS2	16#402#
$FIELD_LAST	50
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FIXED_TYPE
$FLOAT_NAME	NO_SUCH_TYPE
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	75_000.0
$GREATER_THAN_DURATION_BASE_LAST	131_073.0
$GREATER_THAN_FLOAT_BASE_LAST	1.80141E+38
$GREATER_THAN_FLOAT_SAFE_LARGE	1.0E38

$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	1.0E308
$HIGH_PRIORITY	23
$ILLEGAL_EXTERNAL_FILE_NAME1	\NODIRECTORY\FILENAME
$ILLEGAL_EXTERNAL_FILE_NAME2	THIS_FILE_NAME_IS_TOO_LONG_FOR_MY_SYSTEM
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INCLUDE_PRAGMA1	PRAGMA INCLUDE ("A28006D1.ADA")
$INCLUDE_PRAGMA2	PRAGMA INCLUDE ("B28006F1.ADA")
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$INTEGER_LAST_PLUS_1	2147483648
$INTERFACE_LANGUAGE	ASSEMBLER
$LESS_THAN_DURATION	-75_000.0
$LESS_THAN_DURATION_BASE_FIRST	-131_073.0
$LINE_TERMINATOR	ASCII.LF
$LOW_PRIORITY	1
$MACHINE_CODE_STATEMENT	NULL;
$MACHINE_CODE_TYPE	NO_SUCH_TYPE
$MANTISSA_DOC	31
$MAX_DIGITS	9
$MAX_INT	2147483647
$MAX_INT_PLUS_1	2_147_483_648
$MIN_INT	-2147483648
$NAME	NO_SUCH_TYPE_AVAILABLE

MACRO PARAMETERS

$NAME_LIST	MC68020, MC88000, i860, DSP96002
$NAME_SPECIFICATION1	/home/val/acvc1_11/ctest/dev/x2120a
$NAME_SPECIFICATION2	/home/val/acvc1_11/ctest/dev/x2120b
$NAME_SPECIFICATION3	/home/val/acvc1_11/ctest/dev/x2120c
$NEG_BASED_INT	16#F000000E#
$NEW_MEM_SIZE	65535
$NEW_STOR_UNIT	32
$NEW_SYS_NAME	MC88000
$PAGE_TERMINATOR	ASCII.FF
$RECORD_DEFINITION	NEW INTEGER;
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	1024
$TICK	0.000_001
$VARIABLE_ADDRESS	16#0320#
$VARIABLE_ADDRESS1	16#0321#
$VARIABLE_ADDRESS2	16#0322#
$YOUR_PRAGMA	INTERFACE_PACKAGE

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.

LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to linker documentation and not to this report.

2.3.1 Ada Cross-Compiler (ADA96K)

The compiler is activated using the command **ADA96K**.

Syntax: `ADA96K [qualifiers] <source-file-name>`

Parameters: `source-file-name`

Denotes the file containing the source text for the compilation. If no file type is given, .ADA is assumed.

Qualifiers: The qualifiers and options available are detailed in the following table:

Qualifier	Description	Default
/asm_list	Generates an assembler file for each compilation unit. The file name is the unit name with a suffix "_S" or "_B" and the file type is .ASM .	True
/configuration_file	Controls various parameters of the list file layout, input line length, etc.	Internal defaults
/copy_source	Saves the source file for the compilation in the library	False
/list	Generates a list file for the compilation. The file name is that of the source file, with file type .LIS	False
/library	Compiles the unit(s) into the given library	ADA96K_LIBRARY
/optimize	Controls various optimization options	False
/reorder	Activates the code reordering pass of the compiler	False
/suppress_all	Disables generation of code for run-time checks	False
/trace	Enables generation of trace-back information	False
/verbose	Outputs information during compilation	False
/xref	Generates a cross-reference listing in the list file	False

2.3.2 Program Library Manager (PLM96K)

The interactive program library manager is activated using the command PLM96K.

Syntax:

```
PLM96K [/library=<library_name>]
```

Qualifiers:

```
/library=<library_name>
```

This optional qualifier sets the current default library for the PLM96K. If omitted, the default program library defined by the logical name ADA96K_LIBRARY will be opened.

Interactive Commands: The PLM96K commands and options are detailed in the following table:

Command	Description	Default
Create	Creates a new Ada program sublibrary or root library	None
Delete	Deletes the given unit(s) from the current default program library	None
Show	Displays information for the given unit(s) from the current default program library	None
Exit	Leaves the program library manager	None
Help	Displays the PLM96K commands and options	None
Library	Sets the current default program library for the PLM96K	Show the current default library
Type	Displays the Ada source for the given unit(s) from the current default program library	None

Command	Description	Default
Extract	Creates a copy of the stored Ada source for the given unit(s) in the current directory	None
Import	Inserts an externally generated compilation unit body into the current default program library	None
Allocate	Allocates a unit number for an externally generated compilation unit body	None
Verify	Checks the correctness of the current default program library	None

2.3.3 Ada Cross-Linker (LNK96K)

The cross-linker is activated using the command **LNK96K.**

Syntax: LNK96K `[qualifiers]` `<main-unit-name>`

Parameters: **`main-unit-name`**

Denotes the main unit for the Ada program to be linked. The unit given must be a parameterless procedure.

Qualifiers: The qualifier and options available are detailed in the following table:

Qualifier	Description	Default
/library	Specifies the program library where the main and its required units will be looked for	ADA96K_ LIBRARY
/map	Generates a map file for the link. The file name will be the main unit name, with file type .MAP	True

Qualifier	Description	Default
/progress	Outputs information during the link process	False
/log	Generates a log file for the link. The log file name can be given in the command. If not given, the main unit name will be used, with file type .LOG	False
/directives	Uses the directives in the given directives file to guide the link process	None
/target	Specifies the target identifier	096 (ADS)
/with_rts	Includes the required modules from the RTS *in the link*	False
/minimal_ rts	Includes a minimal, single-task RTS in the link	False
/gen_rts	Generates the load file for a new fully-configured RTS executable image	False
/trace	Enables the RTS tracing functions	False
/first_ address	Specifies the initial memory addresses for X, Y and P spaces	(800,800,400)
/last_ address	Specifies the last memory addresses for X, Y, and P spaces	(7FFE,7FFE, FFFE)
/rts_size	Specifies the maximal size of a fully-configured RTS in the different memory spaces	(800,200, 3C00)
/external_ module	Includes the given object file(s) or modules from the given object library in the link	None
/flags	Controls various special functions of the linker	None

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is
    ..........

   type INTEGER is range -2147483648 .. 2147483647;

   type FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;
   type LONG_FLOAT is digits 9 range
                       -1.79769313E+308 .. 1.79769313E+308;

   type DURATION is delta 2**-14 range -131072.0 .. 131072.0;

    ..........
end STANDARD;
```

Appendix F of the Ada Reference Manual

This appendix describes the implementation-dependent characteristics of the AI-ADA/96K Cross-Compilation System, as required in the Appendix F frame of the Ada Reference Manual (ANSI/MIL-STD-1815A).

B.1 Implementation-Dependent Pragmas

The following implementation-dependent pragmas are defined in the compiler:

- suppress_all
- interface_package
- external_subprogram_name

For detailed information on the use of these pragmas, refer to Section 6.8, "Implementation-Dependent Features."

B.2. Implementation-Dependent Attributes

No implementation-dependent attributes are defined for this version.

B.3. Package SYSTEM

The specification of the package SYSTEM:

```
package SYSTEM is

  type ADDRESS  is new INTEGER;
  type PRIORITY is range 1 .. 23;
                   -- Priority 0 is reserved for the Null Task
                   -- Priority 24 is reserved for System Tasks
                   -- Priorities 25..31 are for interrupts
  type NAME      is (MC68020,MC88000,i860,DSP96002);
  SYSTEM_NAME:   constant NAME := DSP96002;
  STORAGE_UNIT:  constant      := 32;
  MEMORY_SIZE:   constant      := 32 * 1024;
  MIN_INT:       constant      := -2_147_483_647-1;
  MAX_INT:       constant      := 2_147_483_647;
  MAX_DIGITS:    constant      := 9;
  MAX_MANTISSA:  constant      := 31;
  FINE_DELTA:   constant      := 2.0 ** (-31);
  TICK:         constant      := 0.000_001;

  type INTERFACE_LANGUAGE is (ASSEMBLER,C,RTS) ;

end SYSTEM;
```

B.4. Representation Clauses

B.4.1 Length Clauses

The following kinds of length clauses are supported:

1. Size specification: T'size

   Supported as described in ARM. For scalar objects residing in the frame, the smallest possible size (in complete words) will always be chosen by the compiler.

2. Specification of a collection size: T'storage_size

   Specifies the number of storage units allocated to the collection associated with access type T.

3. Specification of task size: T'storage_size

   Specifies the number of storage units allocated for each activation of a task of type T. This size includes space for the task's stack, as well as some RTS overhead (approximately 54 words).

4. Specification of small for a fixed point type: T'small

B.4.2. Enumeration Representation Clause

Enumeration representation clauses may specify representations in the range of the predefined type INTEGER.

B.4.3. Record Representation Clause

Record representation clauses are supported as detailed in Section 13.4 of the ARM.

B.5 Implementation-Dependent Names for Implementation-Dependent Components

None defined by the compiler.

B.6 Address Clauses

Address clauses are supported for objects (variables or constants) and task entries (linkage to hardware interrupt); refer to Chapter 16, "AI-TCK Target Configuration Kit."

Address clauses for objects are interpreted as absolute addresses, and code is generated using the ORG directive. The compiler does not check for possible overlaps.

B.7. Unchecked Conversion

No warning is issued when conversion between objects of different sizes is performed. The result of such a conversion is unpredictable.

B.8. Input-Output Packages

Input-Output packages are supplied with the AI-ADA/96K Cross-Compiler System.

Standard_input and Standard_output are supported. External files and file objects are implementation dependent, and therefore are handled as specified in the ARM.

B.8.1. Specification of the Package Sequential_IO

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;
generic
   type ELEMENT_TYPE is private;
package SEQUENTIAL_IO is
   type FILE_TYPE is limited private;
   type FILE_MODE is (IN_FILE, OUT_FILE);

-- File management
procedure CREATE (
   FILE : in out FILE_TYPE;
   MODE : in FILE_MODE  := OUT_FILE;
   NAME : in STRING  := ""; FORM : in   STRING  := "");

procedure OPEN (
   FILE : in out FILE_TYPE;
   MODE : in FILE_MODE;
   NAME : in STRING;
   FORM : in STRING       := "");

procedure CLOSE (FILE : in out FILE_TYPE);
procedure DELETE (FILE : in out FILE_TYPE);
procedure RESET (
   FILE : in out FILE_TYPE;
   MODE : in FILE_MODE);

procedure RESET (FILE : in out FILE_TYPE);
function MODE   (FILE : in FILE_TYPE) return FILE_MODE;
   function NAME   (FILE : in FILE_TYPE) return STRING;

function FORM   (FILE : in FILE_TYPE) return STRING;
function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;
                        -- input and output operations
procedure READ  (
   FILE : in FILE_TYPE;
   ITEM : out ELEMENT_TYPE);

procedure WRITE (
   FILE : in FILE_TYPE;
   ITEM : in ELEMENT_TYPE);
function END_OF_FILE ( FILE : in FILE_TYPE)
     return BOOLEAN;
```

```
 -- exceptions
 STATUS_ERROR : exception renames
    IO_EXCEPTIONS.STATUS_ERROR;
 MODE_ERROR   : exception renames
    IO_EXCEPTIONS.MODE_ERROR;
    NAME_ERROR   : exception renames
    IO_EXCEPTIONS.NAME_ERROR;
    USE_ERROR    : exception renames
    IO_EXCEPTIONS.USE_ERROR;
 DEVICE_ERROR : exception renames
    IO_EXCEPTIONS.DEVICE_ERROR;
 END_ERROR    : exception renames
    IO_EXCEPTIONS.END_ERROR;
 DATA_ERROR   : exception renames
    IO_EXCEPTIONS.DATA_ERROR;

 private
    type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

 end SEQUENTIAL_IO;
```

B.8.2. Specification for Package Direct Input-Output

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;
generic
   type ELEMENT_TYPE is private;
package DIRECT_IO is
   type FILE_TYPE is limited private;
   type FILE_MODE is (IN_FILE, INOUT_FILE, OUT_FILE);
   type COUNT is range 0..INTEGER'LAST;
   subtype POSITIVE_COUNT is COUNT range 1..COUNT'LAST;
   -- File management

procedure CREATE (
   FILE : in out FILE_TYPE;
   MODE : in FILE_MODE  := INOUT_FILE;
   NAME : in STRING     := "";
   FORM : in STRING     := "");

procedure OPEN (
   FILE : in out FILE_TYPE;
   MODE : in FILE_MODE;
   NAME : in STRING;
   FORM : in STRING    := "");

procedure CLOSE (FILE : in out FILE_TYPE);
procedure DELETE (FILE : in out FILE_TYPE);
procedure RESET (
   FILE : in out FILE_TYPE;
   MODE : in FILE_MODE);

procedure RESET (FILE : in out FILE_TYPE);
function MODE   (FILE : in FILE_TYPE) return FILE_MODE;
function NAME    (FILE : in FILE_TYPE) return STRING;
function FORM    (FILE : in FILE_TYPE) return STRING;
function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

-- input and output operations

procedure READ   (
   FILE : in FILE_TYPE;
   ITEM : out ELEMENT_TYPE;
   FROM : in POSITIVE_COUNT);

procedure READ   (
   FILE : in FILE_TYPE;
   ITEM : out ELEMENT_TYPE);

procedure WRITE (
   FILE : in FILE_TYPE;
   ITEM : in ELEMENT_TYPE;
   TO   : in POSITIVE_COUNT);

procedure WRITE (
   FILE : in FILE_TYPE;
   ITEM : in ELEMENT_TYPE);

procedure SET_INDEX(
   FILE : in FILE_TYPE;
   TO   : in POSITIVE_COUNT);

function INDEX( FILE : in FILE_TYPE) return
    POSITIVE_COUNT;

function SIZE (FILE : in FILE_TYPE) return COUNT;
function END_OF_FILE(FILE : in FILE_TYPE)
   return BOOLEAN;

--  exceptions

STATUS_ERROR : exception renames
   IO_EXCEPTIONS.STATUS_ERROR;
MODE_ERROR   : exception renames
   IO_EXCEPTIONS.MODE_ERROR;
NAME_ERROR   : exception renames
   IO_EXCEPTIONS.NAME_ERROR;
USE_ERROR    : exception renames
   IO_EXCEPTIONS.USE_ERROR;
```

```
DEVICE_ERROR : exception renames
   IO_EXCEPTIONS.DEVICE_ERROR;
END_ERROR    : exception renames
   IO_EXCEPTIONS.END_ERROR;
DATA_ERROR   : exception renames
   IO_EXCEPTIONS.DATA_ERROR;

private
   type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end DIRECT_IO;
```

B.8.3. Specification of Package Text Input-Output

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;
package TEXT_IO is
type FILE_TYPE is limited private;
type FILE_MODE is (IN_FILE, OUT_FILE);
type COUNT is range 0 .. INTEGER'LAST;
subtype POSITIVE_COUNT is COUNT range 1 .. COUNT'LAST;

UNBOUNDED: constant COUNT:= 0; -- line and page length
subtype FIELD is INTEGER range 0 .. 35;
subtype NUMBER_BASE is INTEGER range 2 .. 16;
type TYPE_SET Is (LOWER_CASE, UPPER_CASE);

-- File Management

procedure CREATE (
   FILE  : in out FILE_TYPE;
   MODE  : in FILE_MODE := OUT_FILE;
   NAME  : in STRING    := "";
   FORM  : in STRING    := "");

procedure OPEN   (
   FILE  : in out FILE_TYPE;
   MODE  : in FILE_MODE;
   NAME  : in STRING;
   FORM  : in STRING    := "");

procedure CLOSE  (FILE  : in out FILE_TYPE);
procedure DELETE (FILE  : in out FILE_TYPE);
procedure RESET  (
   FILE  : in out FILE_TYPE;
   MODE  : in FILE_MODE);
```

```
  procedure RESET  (FILE  : in out FILE_TYPE);
  function  MODE   (FILE  : in FILE_TYPE) return
                                              FILE_MODE;
  function  NAME   (FILE  : in FILE_TYPE) return STRING;
  function  FORM   (FILE  : in FILE_TYPE) return STRING;
  function  IS_OPEN(FILE  : in FILE_TYPE) return BOOLEAN;

 -- Control of default input and output files

  procedure  SET_INPUT    (FILE : in FILE_TYPE);
  procedure  SET_OUTPUT   (FILE : in FILE_TYPE);
  function   STANDARD_INPUT   return FILE_TYPE;
  function   STANDARD_OUTPUT  return FILE_TYPE;
  function   CURRENT_INPUT    return FILE_TYPE;
  function   CURRENT_OUTPUT   return FILE_TYPE;

 -- specification of line and page lengths

 procedure SET_LINE_LENGTH  (
  FILE : in FILE_TYPE;
  TO   : in COUNT);

 procedure SET_LINE_LENGTH  (TO    : in COUNT);
 procedure SET_PAGE_LENGTH  (
  FILE : in FILE_TYPE;
  TO   : in COUNT);
 procedure SET_PAGE_LENGTH  (TO    : in COUNT);
 function  LINE_LENGTH      (FILE : in FILE_TYPE)
  return COUNT;
 function  LINE_LENGTH return COUNT;
 function  PAGE_LENGTH (FILE : in FILE_TYPE)
     return COUNT;
 function  PAGE_LENGTH return COUNT;

 -- Column, Line, and Page Control

 procedure NEW_LINE (
  FILE    : in FILE_TYPE;
  SPACING : in POSITIVE_COUNT := 1);

 procedure NEW_LINE (SPACING : in POSITIVE_COUNT := 1);

 procedure SKIP_LINE   (
  FILE    : in FILE_TYPE;
  SPACING : in POSITIVE_COUNT := 1);

 procedure SKIP_LINE   (SPACING : in POSITIVE_COUNT
                                              := 1);
 function  END_OF_LINE (FILE : in FILE_TYPE)
  return BOOLEAN;
```

```
function  END_OF_LINE  return BOOLEAN;

procedure NEW_PAGE    (FILE : in FILE_TYPE);
procedure NEW_PAGE                        ;

procedure SKIP_PAGE   (FILE : in FILE_TYPE);
procedure SKIP_PAGE                       ;

function  END_OF_PAGE (FILE : in FILE_TYPE)
 return BOOLEAN;
function  END_OF_PAGE  return BOOLEAN;

function  END_OF_FILE (FILE : in FILE_TYPE)
 return BOOLEAN;
function  END_OF_FILE  return BOOLEAN;

procedure SET_COL (FILE : in FILE_TYPE;
   (TO   : in POSITIVE_COUNT);

procedure SET_COL (TO   : in POSITIVE_COUNT);
procedure SET_LINE (FILE : in FILE_TYPE;
    (TO   : in POSITIVE_COUNT);

procedure SET_LINE  (TO   : in POSITIVE_COUNT);
function  COL       (FILE : in FILE_TYPE)
 return POSITIVE_COUNT;
function  COL   return POSITIVE_COUNT;

function  LINE  (FILE : in FILE_TYPE)
 return POSITIVE_COUNT;
function  LINE   return POSITIVE_COUNT;

function  PAGE  (FILE : in FILE_TYPE)
 return POSITIVE_COUNT;
function  PAGE   return POSITIVE_COUNT;

-- Character Input-Output

procedure GET  (
 FILE : in FILE_TYPE;
 ITEM : out CHARACTER);

procedure GET  (ITEM :  out CHARACTER);

procedure PUT  (
 FILE : in FILE_TYPE;
 ITEM : in CHARACTER);
procedure PUT   (ITEM : in CHARACTER);
```

```
-- String Input-Output

procedure GET   (
 FILE : in FILE_TYPE;
 ITEM : out STRING);

procedure GET  (ITEM : out STRING);

procedure PUT   (
 FILE : in FILE_TYPE;
 ITEM : in STRING);

procedure PUT  (ITEM : in STRING);
procedure GET_LINE   (
 FILE : in FILE_TYPE;
 ITEM : out STRING;
 LAST : out NATURAL);

procedure GET_LINE   (
 ITEM : out STRING;
 LAST : out NATURAL);

procedure PUT_LINE   (
 FILE : in FILE_TYPE;
 ITEM : in STRING);

procedure PUT_LINE   (ITEM : in STRING);

-- Generic Package for Input-Output of Integer Types

generic

 type NUM is range <>;

package INTEGER_IO is
 DEFAULT_WIDTH : FIELD        := NUM'WIDTH;
 DEFAULT_BASE  : NUMBER_BASE  :=        10;

 procedure GET   (
    FILE  : in FILE_TYPE;
    ITEM  : out NUM;
    WIDTH : in FIELD := 0);

 procedure GET   (
    ITEM  :  out NUM;
    WIDTH : in FIELD := 0);
```

```
procedure PUT   (
   FILE  : in FILE_TYPE;
   ITEM  : in NUM;
   WIDTH : in FIELD := DEFAULT_WIDTH;
   BASE  : in NUMBER_BASE := DEFAULT_BASE);

procedure PUT   (
   ITEM  : in NUM;
   WIDTH : in FIELD := DEFAULT_WIDTH;
   BASE  : in NUMBER_BASE := DEFAULT_BASE);

procedure GET   (
   FROM  : in STRING;
   ITEM  : out NUM;
   LAST  : out POSITIVE);

procedure PUT   (
   TO    : out STRING;
   ITEM  : in  NUM;
   BASE  : in  NUMBER_BASE := DEFAULT_BASE);

 end INTEGER_IO;

-- Generic Packages for Input-Output of Real Types

generic

type NUM is digits <>;
package FLOAT_IO is

DEFAULT_FORE : FIELD :=                  2;
DEFAULT_AFT  : FIELD := NUM'digits - 1;
DEFAULT_EXP  : FIELD :=                  3;

procedure GET   (
   FILE   : in FILE_TYPE;
   ITEM   : out NUM;
   WIDTH  : in FIELD := 0);

procedure GET   (
   ITEM   : out NUM;
   WIDTH  : in FIELD := 0);

procedure PUT   (
   FILE : in FILE_TYPE;
   ITEM : in NUM;
   FORE : in FIELD := DEFAULT_FORE;
   AFT  : in FIELD := DEFAULT_AFT;
   EXP  : in FIELD := DEFAULT_EXP);
```

```
procedure PUT  (
   ITEM : in NUM;
   FORE : in FIELD := DEFAULT_FORE;
   AFT  : in FIELD := DEFAULT_AFT;
   EXP  : in FIELD := DEFAULT_EXP);

procedure GET  (
   FROM : in STRING;
   ITEM : out NUM;
   LAST : out POSITIVE);

procedure PUT  (TO : out STRING;
   ITEM : in NUM;
   AFT  : in FIELD := DEFAULT_AFT;
   EXP  : in FIELD := DEFAULT_EXP);

end FLOAT_IO;

generic
type NUM is delta <>;
package FIXED_IO is

DEFAULT_FORE : FIELD := NUM'FORE;
DEFAULT_AFT  : FIELD := NUM'AFT;
DEFAULT_EXP  : FIELD := 0;
procedure GET  (FILE  : in FILE_TYPE;
      ITEM  : out NUM;
      WIDTH : in FIELD := 0);

procedure GET  (
      ITEM  : out NUM;
      WIDTH : in FIELD := 0);

procedure PUT  (
      FILE : in FILE_TYPE;
      ITEM : in NUM;
      FORE : in FIELD := DEFAULT_FORE;
      AFT  : in FIELD := DEFAULT_AFT;
      EXP  : in FIELD := DEFAULT_EXP);

procedure PUT  (ITEM : in NUM;
      FORE : in FIELD := DEFAULT_FORE;
      AFT  : in FIELD := DEFAULT_AFT;
      EXP  : in FIELD := DEFAULT_EXP);

procedure GET (
      FROM : in STRING;
      ITEM : out NUM;
      LAST : out POSITIVE);
```

```
 procedure PUT  (
      TO   : out STRING;
      ITEM : in NUM;
      AFT  : in FIELD := DEFAULT_AFT;
      EXP  : in FIELD := DEFAULT_EXP);

 end FIXED_IO;

 --  Generic package for Input-Output of Enumeration
Types

 generic

 type ENUM is (<>);
 package ENUMERATION_IO is
 DEFAULT_WIDTH   : FIELD    := 0;
 DEFAULT_SETTING : TYPE_SET := UPPER_CASE;
 procedure GET  (
      FILE : in FILE_TYPE;
      ITEM : out ENUM);

 procedure GET  (ITEM : out ENUM);
 procedure PUT  (
      FILE  : in FILE_TYPE;
      ITEM  : in ENUM;
      WIDTH : in FIELD := DEFAULT_WIDTH;
      SET   : in TYPE_SET   := DEFAULT_SETTING);

 procedure PUT  (
      ITEM  : in ENUM;
      WIDTH : in FIELD      := DEFAULT_WIDTH;
      SET   : in TYPE_SET   := DEFAULT_SETTING);

 procedure GET  (FROM : in STRING;
      ITEM : out ENUM;
      LAST : out POSITIVE);

 procedure PUT  (TO : out STRING;
      ITEM : in ENUM;
      SET  : in TYPE_SET := DEFAULT_SETTING);

 end ENUMERATION_IO;
```

```
-- Exceptions

STATUS_ERROR : exception renames
   IO_EXCEPTIONS.STATUS_ERROR;
MODE_ERROR   : exception renames
   IO_EXCEPTIONS.MODE_ERROR;
NAME_ERROR   : exception renames
   IO_EXCEPTIONS.NAME_ERROR;
USE_ERROR    : exception renames
   IO_EXCEPTIONS.USE_ERROR;
DEVICE_ERROR : exception renames
   IO_EXCEPTIONS.DEVICE_ERROR;
END_ERROR    : exception renames
   IO_EXCEPTIONS.END_ERROR;
DATA_ERROR   : exception renames
    IO_EXCEPTIONS.DATA_ERROR;
LAYOUT_ERROR : exception renames
    IO_EXCEPTIONS.LAYOUT_ERROR;

private

   type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end TEXT_IO;
```